1. Tempo disponibile 120 minuti.
2. Non è possibile consultare appunti, slide, libri, persone, siti web, ecc.
3. Scrivere in modo leggibile, su ogni foglio, nome, cognome e numero di matricola.
4. Le soluzioni agli esercizi che richiedono di progettare un algoritmo devono:
   - spiegare a parole l'algoritmo (se utile, anche con l'aiuto di esempi o disegni),
   - fornire e commentare lo pseudo-codice (indicando il significato delle variabili),
   - calcolare la complessità (con tutti i passaggi matematici necessari),
   - se l'esercizio ammette più soluzioni, a soluzioni computazionalmente più efficienti e/o concettualmente più semplici sono assegnati punteggi maggiori.

IMPORTANTE: risolvere l'esercizio 1 su foglio separato. Infatti, al termine la consegna dell'esercizio 1 deve essere fatta separatamente rispetto alla consegna degli esercizi 2–3–4.

1. Calcolare la complessità $T(n)$ del seguente algoritmo MYSTERY1:

Algorithm 1: MYSTERY1(INT n) $\rightarrow$ INT

```
x = 0
res = 1
while n ≥ 1 do
    n = n/3
    x = x + 1
    res = res × 2×MYSTERY2(2x)
return res

function MYSTERY2(INT m) → INT
if m ≤ 1 then
    return 1
else
    x = 1
    for i = 1 to m do
        x = 2 × x
    return MYSTERY2(⌊2m/3⌋) + x
```

2. Si consideri il seguente array di numeri $A[8, 3, 9, 2, 6, 5, 4, 7, 1]$. Indicare come cambia il contenuto dell'array dopo l'esecuzione dell'operazione *heapify* utilizzata per trasformarlo in un *max–heap binario*. Su tale *max–heap binario* viene poi eseguita l'operazione *deleteMax*: mostrare come cambia il contenuto dell'array dopo l'esecuzione di tale operazione. Considerate le implementazioni di *heapify* e *deleteMax* descritte a lezione (e riportate nelle slide del corso). Consiglio: considerate l'esecuzione delle operazioni sulla rappresentazione dell'array come albero binario, e poi riportate in formato array l'albero ottenuto dopo l'esecuzione delle operazioni.
3. Bisogna riempire completamente una cisterna di capacità C versandoci il contenuto di alcuni fra n contenitori chiusi ermeticamente, facendo attenzione a non aprire più di K contenitori. Progettare un algoritmo che data la capacità C della cisterna (espressa come numero reale positivo), il numero massimo K di contenitori utilizzabili (espresso come numero intero non negativo), e l'array $A[1..n]$ dove $A[i]$ indica la quantità di liquido nel contenitore i-esimo (espressa come numero reale positivo), verifica se è possibile riempire completamente la cisterna usando al più K contenitori. Si noti che i valori K ed n sono indipendenti, e quindi possono appartenere ad ordini di grandezza diversi.

4. Considerate un grafo orientato $G = (V, E)$ tale che per ogni $v \in V$ esiste un campo $v.nat$ che contiene un numero naturale (ovvero, un intero positivo). Progettare un algoritmo che dato G e due vertici $s \in V$ e $t \in V$, stampa (se esiste) il cammino $v_0, v_1, v_2, \ldots, v_k$ da s in t (ovvero, $v_0 = s$ e $v_k = t$) che minimizza la somma dei numeri associati ai nodi del cammino (ovvero, minimizza $\sum_{i=0}^{k} v_i.nat$).

The End.